SPICE TO VERILOG NETLIST TRANSLATOR

Background of the Invention

5   a.   Field of the Invention

The present invention pertains to integrated circuit development software and more specifically to translating a circuit description employing a SPICE format to that of a Verilog format.

10   b.   Description of the Background

Integrated circuit design may be custom or semi-custom. Semi-custom may employ gate array or standard cell methodologies. Gate array employs a base set of functions fabricated on a semiconductor die that are later interconnected to reflect the function of the design. Interconnection may employ metal, polysilicon or other layers.

15   Standard cell employs a predefined set of cells exhibiting a common dimension that may be placed in rows and blocks, the order determined by functions to be implemented and routing of interconnect between cells or groups of cells. Verilog, a hardware description language, is frequently employed in the design process. Verilog may be used to specify the initial design, to provide input to synthesis tools, and to check post layout operation.

20   At times, the predefined set of cells of a standard cell library may not provide a desired function, or may not provide the speed, size, or power consumption desired. In these circumstances, new cells may be created, or a custom block of logic incorporating the desired function may be designed. The design of the custom block of logic may employ SPICE (Special Programs for Interactive Circuit Elements) to specify and simulate the

25   design. SPICE supports both logical and timing simulation. In order to simulate the whole design that comprises Verilog and SPICE defined elements, Verilog must be converted to SPICE or SPICE must be converted to Verilog. A number of simulation tools require a SPICE format input and tools are provided to convert Verilog to SPICE such that the whole design may be simulated using SPICE. There are not tools that

30   provide SPICE to Verilog conversion, as may be employed to verify Verilog to SPICE conversion, or to allow simulation of a design in Verilog for those simulation tools that

support Verilog formats. Further, changes made to SPICE designs are not easily verified. Therefore a new method to convert a SPICE netlist to a Verilog netlist is needed.

5   **Summary of the Invention**

The present invention overcomes the disadvantages and limitations of the prior art by providing a system and method of translating a SPICE netlist to Verilog. By mapping SPICE constructs to Verilog constructs, and by mapping SPICE syntax to Verilog syntax,

10   the present invention provides an easy to use, automatic conversion program. Advantageously, the present invention allows SPICE based designs to be converted to Verilog to confirm the accuracy of prior conversion from Verilog to SPICE. SPICE may be required as a simulation input format for some simulation tools. Such tools may be employed to simulate circuits comprising semi-custom and custom components, and may

15   also be employed to simulate circuits comprising analog and digital circuitry, as may be employed in communication systems.

The present invention therefore may comprise a method for designing an integrated circuit comprising: partitioning a design into a plurality of function blocks; designing a first one of the plurality of function blocks employing Verilog to produce a

20   first block design; designing a second one of the plurality of function blocks employing SPICE to produce a second block design; converting the first block design to SPICE to produce a converted first block design; simulating operation of the converted first block design and the second block design; and translating the converted first block design to Verilog to produce a translated first block design.

25   The present invention provides flexibility in the format of simulation such that SPICE designs may be converted to Verilog and simulated in Verilog. Such simulation may comprise testing and fault coverage analysis. Further, a SPICE file that is converted to Verilog may be used as stimulus to another block that is being designed, allowing design and simulation to occur in a single format. Advantageously, the circuit and signal

30   names of the SPICE file may be retained, or may be translated to a form readily associable with the original names such that translated files may easily be compared with

2

the SPICE source file. The hierarchy of the SPICE description may also be maintained, allowing simplified review of circuit structure

The invention therefore may further comprise a method for translating a SPICE netlist to Verilog comprising: opening a SPICE file; translating ".SUBCKT"

5   instantiations to "module"; translating ".ENDS" statements to "endmodule"; translating SPICE circuit elements to Verilog format; and removing discrete circuit elements.

The present advantage provides the flexibility of format that allows developers to employ a wider range of simulation tools while maintaining verification of the converted design. The method of the present invention may be employed in a batch process,

10   providing automatic conversion. The aforementioned flexibility of format may allow greater utilization of resources by providing a choice to simulate using SPICE or to simulate using Verilog.

15   __Description of the Figures__

In the figures,

Figure 1 is an overview of a design process. 𝒫 ᵃ ˙

Figure 2 depicts Verilog to SPICE conversion and combined design simulation using SPICE.

20   Figure 3 depicts Verilog to SPICE conversion, SPICE to Verilog translation for verification, and combined design simulation using SPICE.

Figure 4 depicts SPICE to Verilog translation and combined design simulation using Verilog.

Figure 5 is a flowchart of a SPICE to Verilog translation process.

25

                        __Detailed Description of the Invention__

Figure 1 is an overview of an integrated circuit design process. Process 100

30   begins at step 102 where a function specification for a design is received. At step 104, the design may be partitioned into functional blocks. At step 106, logic design of the blocks

3

partitioned in step 104 is performed. Logic design may employ pre-defined circuits such as counters and registers that are provided with a design library. Logic design may employ a hardware descriptive language such as Verilog. Verilog is a Hardware Description Language; a textual format for describing electronic circuits and systems.

5   Applied to electronic design, Verilog may be used for verification through simulation, for timing analysis, for test analysis (testability analysis and fault grading) and for logic synthesis. Verilog HDL is an IEEE standard - number 1364. Through Verilog, the designer may select circuit elements and define the interconnection between elements. At step 106, it may be determined that custom design is desired to meet a design goal such

10   as speed, power, circuit size, or other parameter. Custom design may be performed at step 108 and may employ software simulation tools such as SPICE. Once a design for one block or a plurality of blocks is completed, the process advances to step 110 where the design is simulated. At step 110, Verilog files may be converted to SPICE to allow simulation with custom logic. Simulation may employ a stimulus file that provides a

15   sequence of input signals, allowing the designer to check operation of the block or blocks. The steps of block design 106, custom design 108, and simulation 110 may be reiterative in that a design may be altered to correct errors or to achieve implementation goals. At step 112, the design may be synthesized wherein the circuit description provided through Verilog or other hardware descriptive language is converted to logic

20   gates. The logic gates may be selected from a logic library such as a standard cell library. For each type of function defined in the design, such as AND, OR and other gates, there are commonly a plurality of versions of each function wherein versions may vary in speed, power, output drive, or other characteristics. Post synthesis simulation 114 may employ SPICE values for custom logic and may employ parameters associated with the

25   logic gates selected in step 112 to provide a more accurate estimation of performance. The step 114 simulation may include comparison with design goals and may result in repetition of previous steps to meet design goals. The process then continues at step 116 where routing and physical layout of the design is performed. Post layout simulations (not depicted) may be performed to confirm correct design operation. The process ends at

30   step 118.

Figure 2 depicts Verilog to SPICE conversion and combined design simulation using SPICE. Verilog design 202 is input to Verilog to SPICE converter 204. Verilog to SPICE converter 204 may employ software such as v2lvs from Mentor Graphics Corporation that has headquarters at 8005 SW Boeckman Road, Wilsonville, OR 97070.

5 Custom design 206, in a SPICE format, may be combined with the output of Verilog to SPICE converter 204 to produce combined design 208 that then may be simulated in SPICE format at step 210. Simulation 210 may employ programs such as Timemill from Synopsys Corporation, headquartered at 700 East Middlefield Rd. Mountain View, CA 94043. Simulation 210 may comprise analog, digital or mixed signal simulation.

10 Figure 3 depicts Verilog to SPICE conversion, SPICE to Verilog translation for verification, and combined design simulation using SPICE. Verilog design 302 is input to Verilog to SPICE converter 304. Custom design 310, in a SPICE format, may be combined with the output of Verilog to SPICE converter 304 to produce combined design 312 that may be simulated in SPICE format at step 314. The output of Verilog to SPICE

15 converter 304 may be input to SPICE to Verilog translator 306. Verilog design 302 and the output of SPICE to Verilog translator 306 are supplied to conversion verification 308 where the accuracy of Verilog to SPICE conversion may be checked.

Figure 4 depicts SPICE to Verilog translation and combined design simulation using Verilog. Custom logic 404, in SPICE format, is input to SPICE to Verilog

20 translator 406. Verilog design 402 is combined with the output from SPICE to Verilog translator 406 to produce combined design 408 that may be simulated in Verilog format at step 410.

Figure 5 is a flowchart of a SPICE to Verilog translation process. The translation process 500 begins with SPICE netlist 502 that is opened and read in at step 504. Table 1

25 below lists an example SPICE file input and Table 2 below lists an example Verilog file output. At step 506, instantiations of the heading "SUBCKT" are translated to "module". A circuit name, and input and output signal names, may follow the "SUBCKT" heading. The circuit name may be identified as the first work following "SUBCKT" and may be utilized as the module name. Input and output signal names follow the circuit name and

30 may be may be employed in the Verilog output file. For example, the "SUBCKT INV_CHAIN Z A" line of table 1 may be employed to generate the "module

5

INV_CHAIN (Z, A)" entry listed in table 2. Retaining the circuit name and input and output signal names allows simplified association of the SPICE input file with the Verilog output file. Alternately, new circuit and signal names may be introduced in the translation process. At step 508, "ENDS" statements are translated to "endmodule". At

5   step 510, "x" elements are translated to Verilog format. The 'x' elements may comprise an instance name, signal nodes, and a circuit element descriptor. For example, the "XINV3 2 3 INV" line listed in table 1 may be translated to the "INV INV3 (.Z(3), .A(2)):" line listed in table 2. The translation of instance names ("XINV1" to "INV1" in the above example), may employ simple truncation of a leading "x" in element names, or may

10 employ renaming. Renaming may be total or in part and may employ a lookup table, an algorithm, or rules to define the instance name. Advantageously, the present invention allows the hierarchy of the SPICE design and signal name associations to be maintained. The signal node names of the SPICE file may be employed to define wires in Verilog, and to specify signal connections to Verilog circuit elements. As may be observed in

15 table 2, the signal nodes listed in table 1 are defined as wires. The circuit element descriptor may be translated to Verilog syntax. IN the examples of tables 1 and 2, INV is used for both SPICE and Verilog syntax. At step 512, discrete circuit elements, such as resistors, capacitors, transistors, and inductors, for example, are removed. At step 514, the modified file is written out to produce Verilog structured netlist 516.

20

Table 1 - EXAMPLE SPICE INPUT

```
.SUBCKT  INV_CHAIN  Z  A
XINV1  1  A  INV
XINV2  2  1  INV
XINV3  3  2  INV
XINV4  4  3  INV
XINV5  Z  4  INV
.ENDS

.SUBCKT  INV  Z  A
MP1  Z  A  VDD  VDD  W=1U   L=0.25U
MN1  Z  A  VSS  VSS  W=0.5U  L=0.25U
.ENDS
```

25

30

35

Table 2 - EXAMPLE VERILOG OUTPUT

```
module INV_CHAIN (Z, A);
output Z;
input A;
wire 1;
wire 2;
wire 3;
wire 4;

INV INV1 (.Z(1), .A(A));
INV INV2 (.Z(2), .A(1));
INV INV3 (.Z(3), .A(2));
INV INV4 (.Z(4), .A(3));
INV INV5 (.Z(Z), .A(4));

endmodule
```

The above example is illustrative of one embodiment of the present invention. A Perl code listing of an embodiment of the invention is shown in Appendix A. References to SPICE include any hardware description language compliant with IEEE SPICE standards. The above description need not be implemented in the exact order described. Other methods may be employed to achieve equivalent results. Naming conventions may be employed to simplify the translation process. For example, SPICE files may be formatted such that only input signals begin with the letter "I" and that only output signals begin with the letter "O". Additionally, signals may be more easily identified through predefined delimiters. For example, a plus sign may be used to delimit and indicate input signals and a minus sign may be used to delimit and indicate output signals.

As described, the present invention provides a quick and convenient method that allows a designer to verify the accuracy of conversion of Verilog designs to SPICE, and to convert SPICE designs to Verilog for simulation using Verilog. As previously noted, the translation provided by the present invention also provides greater flexibility in the simulation tools that may be employed.

The foregoing description of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed, and other modifications and variations may be possible in light in the above teachings. The embodiment was chosen and described in order to best explain the principles of the invention and its practical application to thereby enable

others skilled in the art to best utilize the invention in various embodiments and various modifications as are suited to the particular use contemplated. It is intended that the appended claims be construed to include other alternative embodiments of the invention except insofar as limited by the prior art.

5

APPENDIX A

```perl
#!/usr/local/bin/perl

# Copyright (c) 2001
# LSI Logic, Inc.
#
# Andy Rankin
# Ph: (970) 206-5107
# email: arankin@lsil.com
#
# sp2vlog.pl
#
# Usage: sp2vlog.pl -s spfile [-d dfile] [-h]
#
#          -s <spfile>    Spice transistor level file to be
translated (required)
#          -d <dfile>     Data file from vlog2sp.pl containing port
types (defaults to basename.data)
#          -h                 prints this message
#


# ADD CLEANER TO THIS FILE TO GET + SIGNS REMOVED
# ADD SOMETHING TO VLOG2SP TO MAKE SPICE COMPATIBLE LINE LENGTHS


###############
# Set Defaults #
###############

$tech="gflx";
$subckt_lead_string="x";


###############################
# Parse Command Line Arguments #
###############################

require "newgetopt.pl";
&NGetOpt("s:s","d:s","h");

if ($opt_h) { &usage; }
if ($opt_s) {
     $spfile=$opt_s;
}
else {
     print("\nERROR: Must specify spice file\n");
     &usage;
}

$basename=`basename $spfile .sp`;
chop($basename);
```

14

```
     if ($opt_d) { $dfile=$opt_d; }
     else { $dfile=$basename.".data"; }


     ##############################
 5   # Open spice and data files #
     ##############################

     open(INFILE,"$spfile") || die("Unable to open $spfile for
     reading: $!\n\n");
10   @infile=<INFILE>;
     close(INFILE);

     open(INFILE,"$dfile") || die("Unable to open $dfile for reading:
     $!\n\n");
15   @dfile=<INFILE>;
     close(INFILE);


     foreach (@infile) {
           @pin_list=();
20         tr/A-Z/a-z/;
           if (/^\s*.subckt\s/) {
                 # If the line is a spice subckt definition, it is the
     same as a module definition in
                 # verilog. Split the line to extract the subcircuit
25   name and pins:
                 chop;
                 ($foo,$module,@subckt_pins)=split(/\s+/);
                 @module_pins=&remove_busses(@subckt_pins); # remove
     bus notation from subcircuit pins
30               $pinlist=join(", ",@module_pins);
                 print ("module $module \( $pinlist \)\;\n"); # print
     the verilog module definition line

                 # For each pin on the module, determine the port type
35   from the data file:
                 foreach $pin (@module_pins) {
                       foreach $port (@dfile) {
                             if ($port =~ /$module $pin (\S+)/) {
     $port_type=$1; last; }
40                                 else { $port_type=""; }
                       }
                       @indices=&consolidate_bus($pin,@subckt_pins); #
     If it's a bus, add bus notation

45                     # Print the input/output definitions with bus
     notation as necessary:
                       if ($port_type) {
                             if (@indices) { print("$port_type
     [$indices[$#indices]:$indices[0]] $pin\;\n"); }
50                             else { print("$port_type $pin\;\n"); }
                       }
```

15

```perl
            }

            # Print wire definitions for all the internal signals:
            foreach (@dfile) {
                    if (/$module (\S+) wire/) { print("wire $1\;\n");
    }
                }

            }
    elsif (/^\s*.ends\s*/) {
            print("endmodule\n\n");
    }
    elsif (/^\s*x/) {
            s/^\s+//;
            s/\s+$//;
            @signal_names=split(/\s+/);
            $instance_name=shift(@signal_names);        # Shift the
instance name from front of line
            $instance_name=~s/^$subckt_lead_string//; # Remove the
X from the
            $subckt_name=pop(@signal_names);            # Pop the
module name from the end of line
            @signal_names=&clean_signals(@signal_names);

            #grep the spice file for the subcircuit definition,
and extract the pin names:
            ($foo,$foo2,@subckt_pins)=split(/\s+/,`grep -i
".SUBCKT $subckt_name " $spfile`);

            # Verify that the number of signals in the spice
subcircuit call matches the number of
            # pins in the spice subcircuit definition:
            if ($#signal_names != $#subckt_pins) {
                    die("ERROR: $instance_name $subckt_name number of
pins do not match\n\n");
            }

            # Pair up the signal names in the spice subcircuit
call with the pin names
            # on the spice subcircuit definition:
            for($i=0;$i<=$#signal_names;$i++) {
                    $subckt_pins[$i]=~tr/A-Z/a-z/;

        push(@pin_list,".$subckt_pins[$i]\($signal_names[$i]\)");
            }

            # Join the list of signal/pin pairs and print it with
the module instantiation:
            $pin_string=join(", ",@pin_list);
            print("$subckt_name $instance_name ( $pin_string
)\;\n");
```

16

```perl
        }


    }
sub usage {

        # This subroutine prints usage information

        print("\nusage: sp2vlog.pl -s spfile [-d dfile] [-h]\n\n");
        print("\t-s <spfile>\tspice transistor level file to be
translated (required)\n");
        print("\t-d <dfile>\tdata file from vlog2sp.pl containing
port types (defaults to basename.data)\n");
        print("\t-h\t\tprints this message\n");
        die("\n");
}

sub consolidate_bus {

        # This subroutine finds the indices for a given pin, sorts
them, and returns them.

        my($pin,@pins) = @_;
        my(@indices);

        foreach (@pins) {
            if (/$pin\[(\d+)\]/) {
                push(@indices,$1);
            }
        }
        @indices=sort(@indices);

    }

sub remove_busses {

        # This subroutine removes bus notation from the pins on the
spice subcircuit line.
        # It includes each bus as one pin in the pinlist. It
returns the pin list.

        my(@pinlist) = @_;
        my(@newpinlist);

        foreach (@pinlist) {
            s/\[\d+\]//g;
            if(&element_exists($_,@newpinlist)) {}
            else { push(@newpinlist,$_); }
        }
```

```perl
        return(@newpinlist);
}

sub element_exists {

        # This subroutine checks to see if an element exists in an
array.

        my($element,@array)=@_;

        foreach (@array) {
            if ($_ eq $element) { return(1); }
        }

        return(0);
}

sub clean_signals {

        # This subroutine converts vss and vdd signals to verilog
logic 0 and logic 1. It also
        # removes dummy pins.

        my(@signals) = @_;

        foreach (@signals) {
            if ($_ eq "vss") { $_ = "1'b0"; }
            if ($_ eq "vdd") { $_ = "1'b1"; }
            if (/^dummy/) { $_ = ""; }
        }
        @signals;
}
```